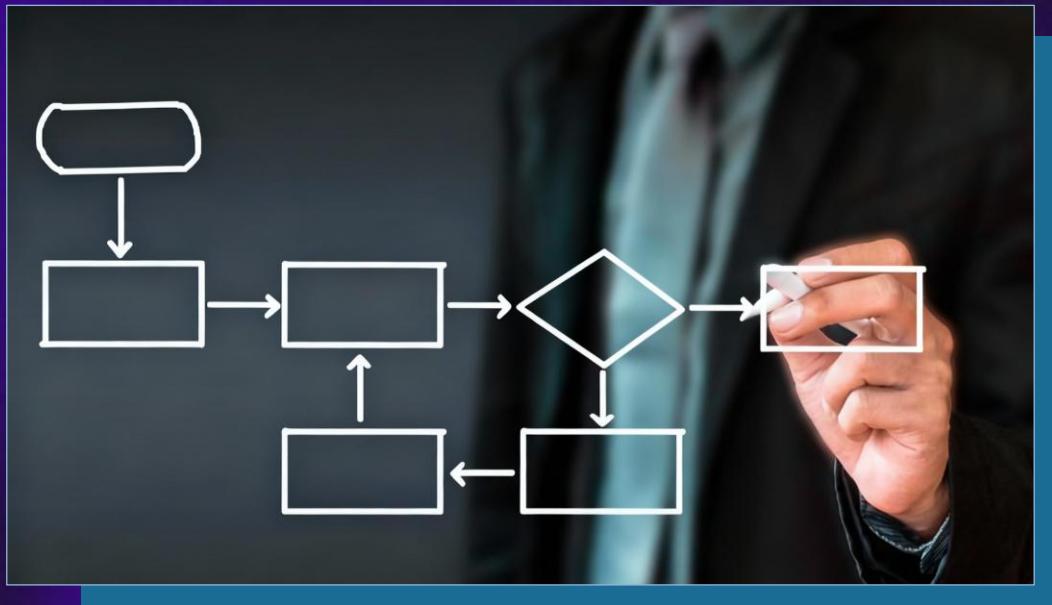


Control-Flow Logic

Conditions, If-Else, Switch-Case, Code Blocks,
Loops (for, while, do-while), Nested Loops



Svetlin Nakov, PhD
Co-founder @ SoftUni



SoftUni AI

<https://ai.softuni.bg>

Agenda

1. Comparison Operators: `==`, `>`, `<`, etc.
2. Conditional Statements
 - **if-else**, Series of **if-else**, **switch-case**
 - Nested **if-else**, Complex Conditions with `&&`, `||` and `!`
3. Loops
 - **for** Loops, Loops with a Step
 - **while** Loops, **do-while** Loops, Infinite Loops
 - Nested Loops: Loops inside Loops



Sli.do Code

#AI-Programming

Join at

slido.com

#AI-Programming



Breaks

20:00 / 21:00



Comparison Operators

Comparing Values: ==, >, <, etc.

JAVASCRIPT

COMPARISON OPERATORS

==

<

<=

==

!=

>

>=

!=

Comparison Operators in JS



| Action | Operator |
|--------------------------------|--------------------|
| Equal values | <code>==</code> |
| Equal values and data types | <code>===</code> |
| Different values | <code>!=</code> |
| Different values or data types | <code>!==</code> |
| Greater than | <code>></code> |
| Less than | <code><</code> |
| Greater than or equal | <code>>=</code> |
| Less than or equal | <code><=</code> |

Comparing Values

- Comparing values returns a Boolean result → true / false

```
let a = 5;  
let b = 10;  
console.log(a < b);          // true  
console.log(a > 0);          // true  
console.log(a > 100);         // false  
console.log(a < a);          // false  
console.log(a <= 5);          // true  
console.log(b == 2 * a);       // true  
console.log("2" === 2);        // false
```



Boolean Variables

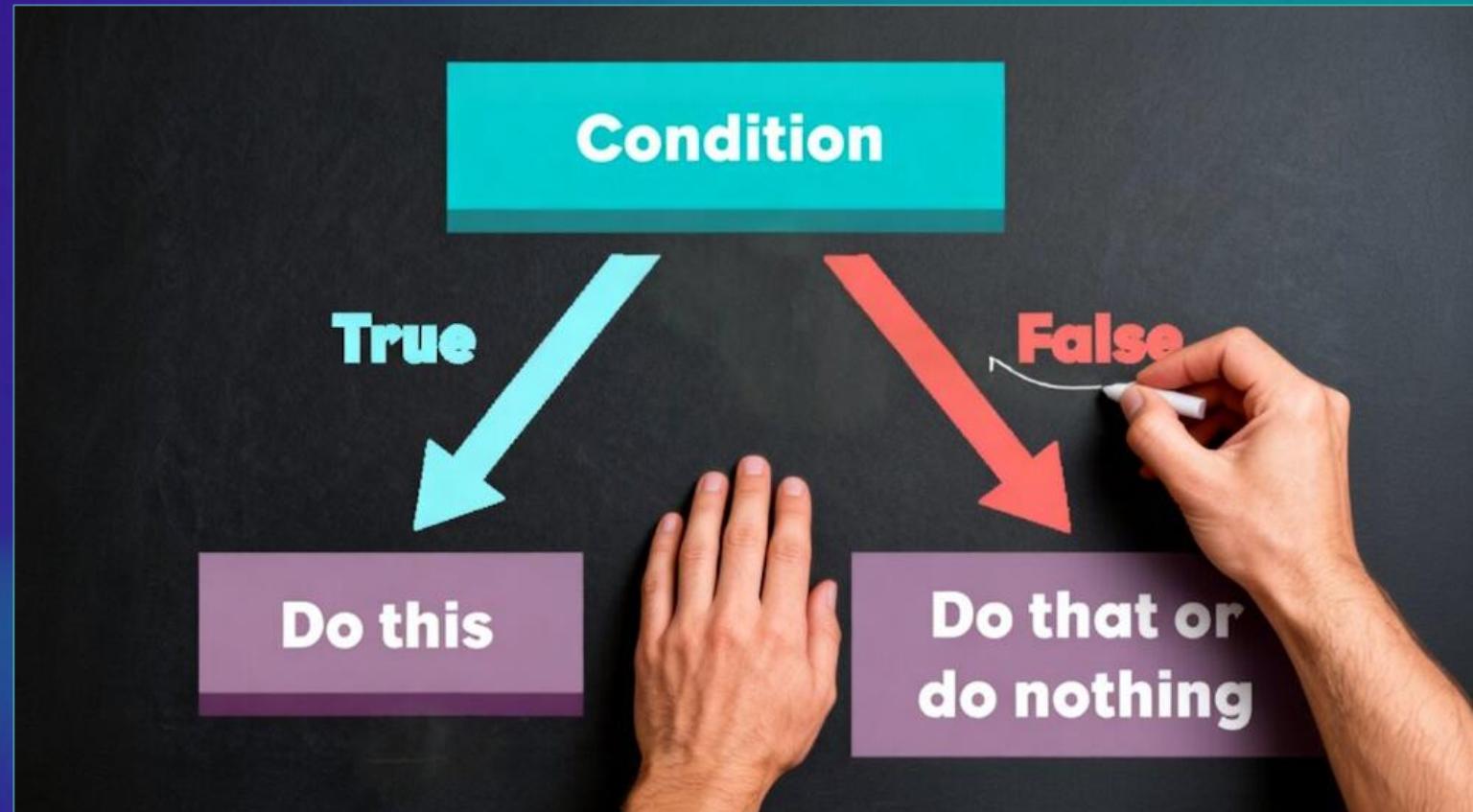
- The **result of comparison** can be hold in a **variable**

```
let a = 5;  
let isPositive = a > 0;  
console.log(isPositive); // true
```

```
let a = -5;  
let isPositive = a >= 0;  
console.log(isPositive); // false
```

Conditional Statements

Using if and if-else



If-Statement

- The **if-statement** checks a condition and does something when the condition is true

Condition (Boolean expression)

```
if (condition) {  
    // code to execute ...  
    // code to execute ...  
}
```

Code to execute when the condition is met

- Example:

```
let grade = 6.00;  
if (grade >= 5.50) {  
    console.log("Excellent");  
}
```

If-Statement – More Examples

```
let temperature = 0.5;  
console.log(`Temperature is ${temperature} °C`);  
  
if (temperature < 10) {  
    console.log("It's cold outside.");  
    console.log("Dress well.");  
}  
  
if (temperature > 30)  
    console.log("It's hot outside.");  
  
console.log("Learn more at https://weather.com");
```

Code block:
{ ... }

Single line:
no code block

If-Else Statement

- The **if-else-statement** does different things depending on a Boolean condition

```
if (condition) {  
    // some code ...  
    // some code ...  
}  
else {  
    // another code ...  
}
```

```
let temperature = 25;  
if (temperature >= 30) {  
    console.log("Hot");  
}  
else {  
    console.log("Not hot");  
}
```

Problem: Odd or Oven

- Write a **function** to check is given **number** is **odd** or **even**
- Sample usage:

```
console.log(oddOrEven(3));  
// odd
```



```
console.log(oddOrEven(12));  
// even
```



```
console.log(oddOrEven(-2));  
// even
```



Solution: Odd or Oven

- Use a simple **if-else** statement:

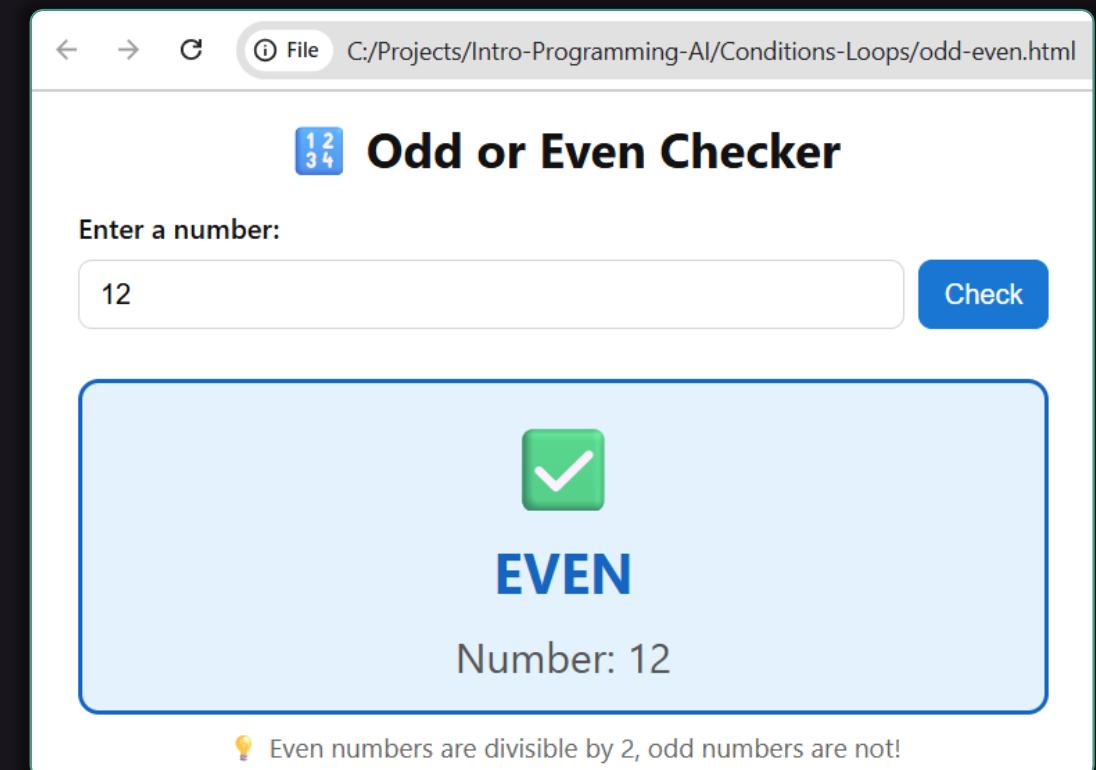
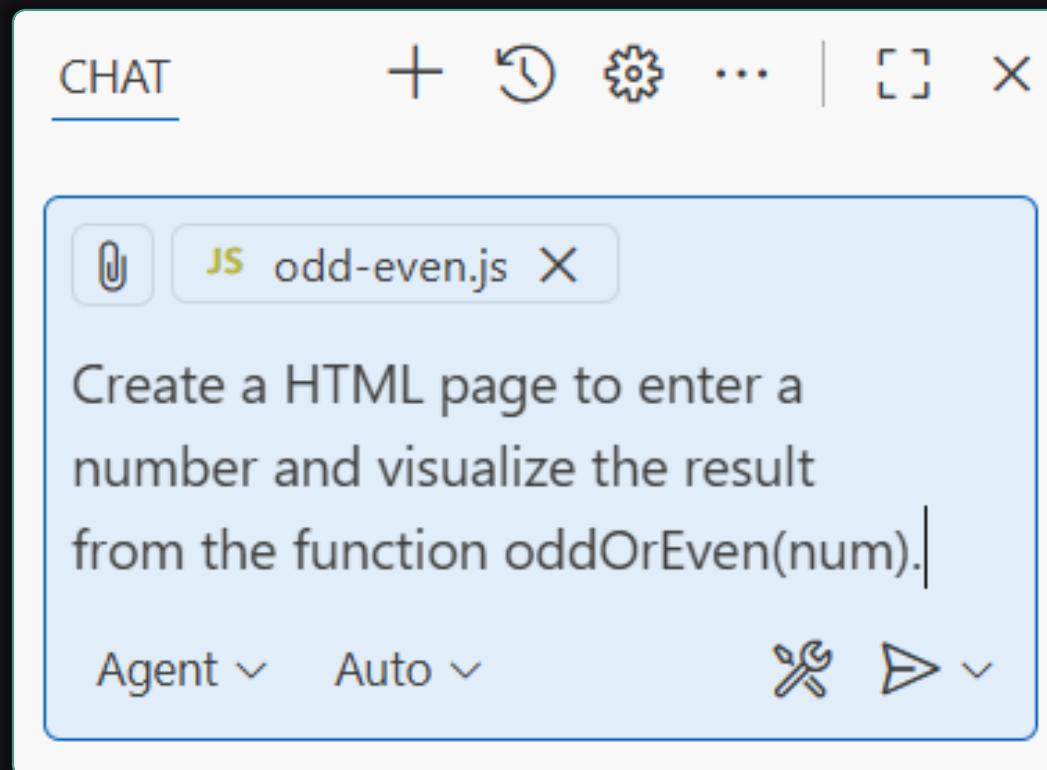
```
function oddOrEven(num) {  
    if (num % 2 == 0)  
        return "even";  
    else  
        return "odd";  
}
```

```
function oddOrEven(num) {  
    // Use the ?: operator  
    let res = (num % 2 == 0)  
        ? "even" : "odd";  
    return res;  
}
```

- Judge link: <https://alpha.judge.softuni.org/contests/control-flow-logic/5271>

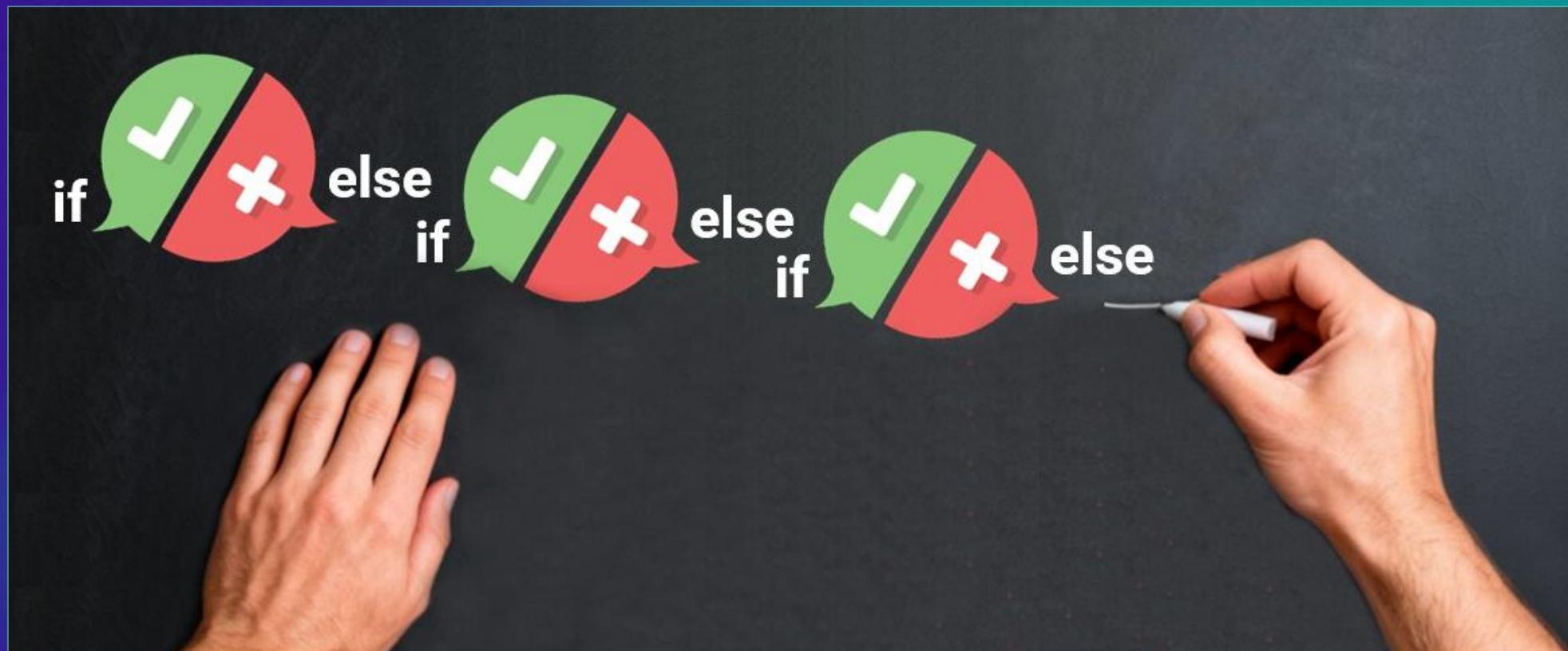
Visualize the Function with HTML

- A simple GitHub Copilot **prompt** will generate an **UI to visualize the function**:



Series of If-Else

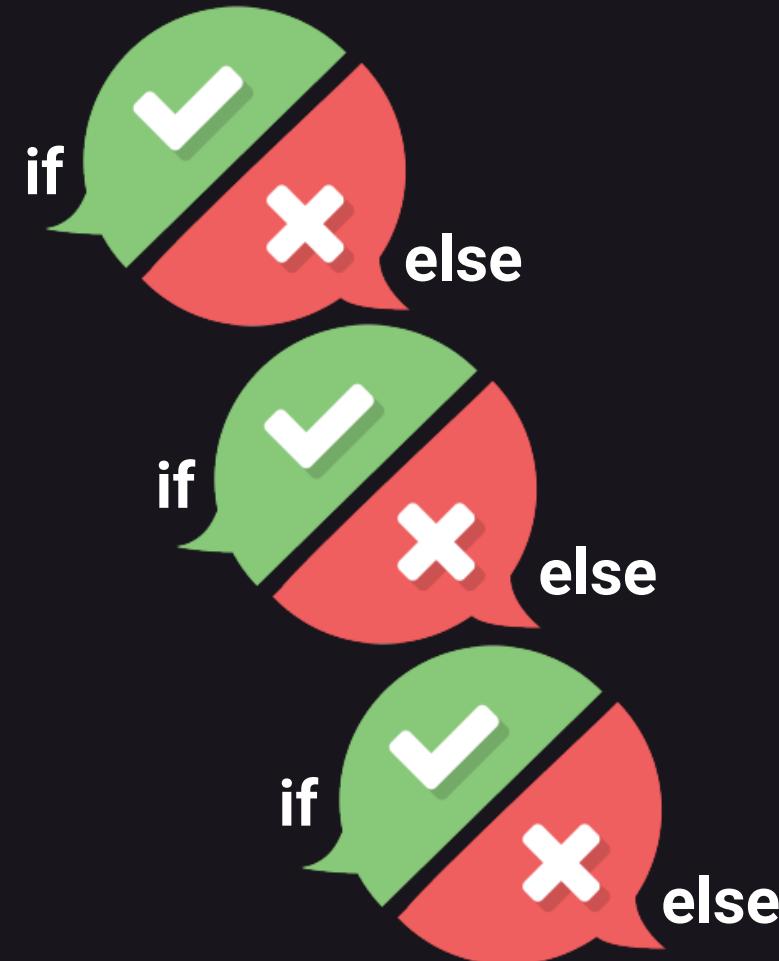
Using if-else-if-else-... in Series



Series of If-Else

- We can use multiple **if-else-if-else** as series:

```
let speed = 20;  
if (speed >= 100)  
    console.log("Fast")  
else if (speed >= 40)  
    console.log("Average")  
else if (speed > 0)  
    console.log("Slow")  
else  
    console.log("Invalid");
```



Problem: Speed Evaluation

- Write a **function** to check if given **speed** is **fast**
 - **Speed ≥ 100 km/h** is considered **fast**
 - $40 \leq \text{speed} < 100$ km/h → considered **average**
 - $0 < \text{speed} < 40$ km/h → considered **slow**
 - **Speed ≤ 0 km/h** is considered **invalid**

```
console.log(evaluateSpeed(120)); // fast
console.log(evaluateSpeed(99)); // average
console.log(evaluateSpeed(5)); // slow
console.log(evaluateSpeed(-1)); // invalid
```

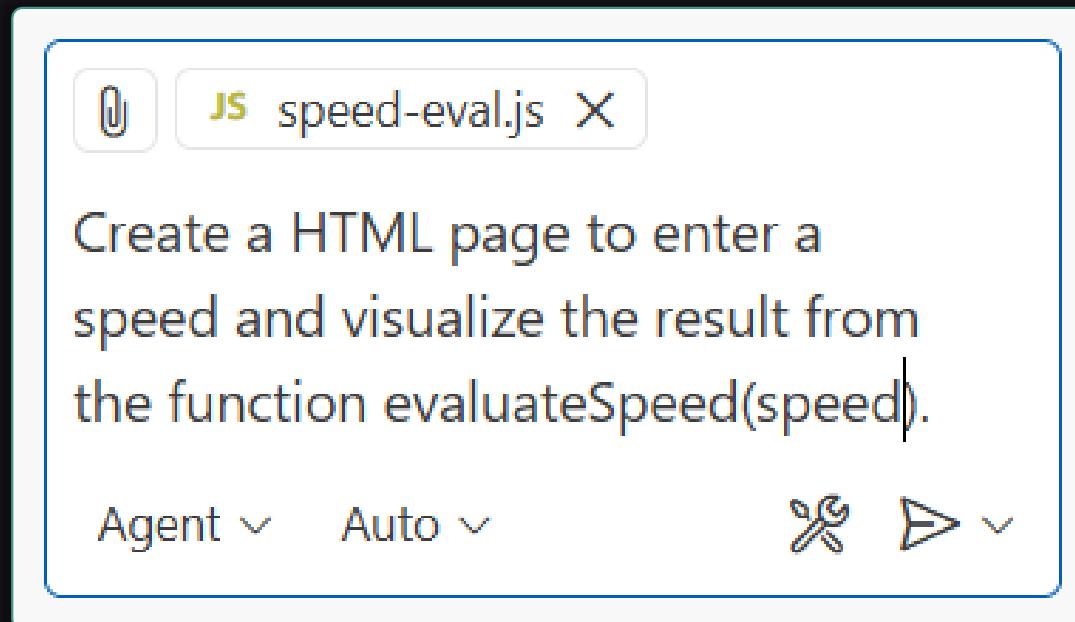
Solution: Speed Evaluation

```
function evaluateSpeed(speed) {  
    if (speed >= 100)  
        return "fast";  
    else if (speed >= 40)  
        return "average";  
    else if (speed > 0)  
        return "slow";  
    else  
        return "invalid";  
}
```

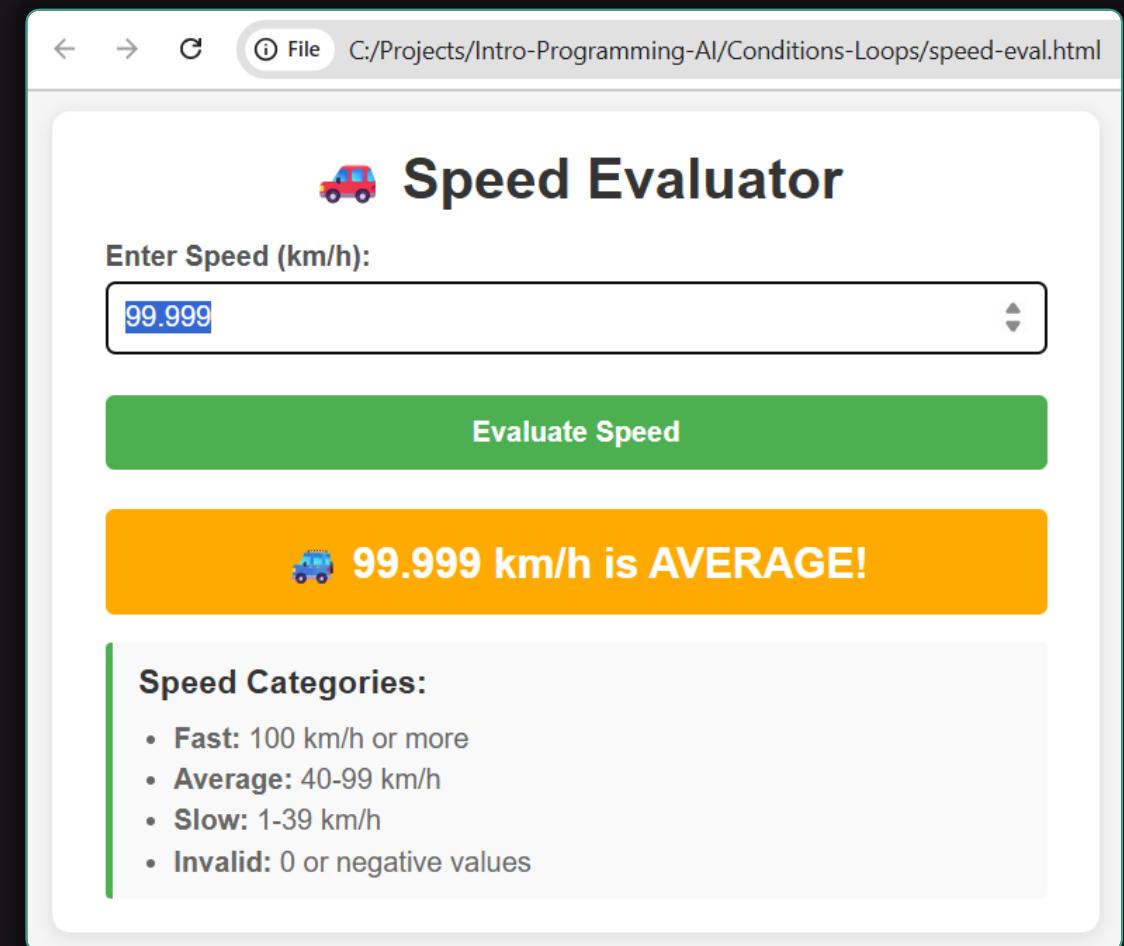


Visualize the Function with HTML

- A simple GitHub Copilot **prompt** will generate an **UI to visualize the function**:



The screenshot shows the GitHub Copilot interface. A code suggestion for a file named "speed-eval.js" is displayed. The suggestion text reads: "Create a HTML page to enter a speed and visualize the result from the function evaluateSpeed(speed).". Below the suggestion, there are two dropdown menus: "Agent" and "Auto", and a set of icons for interacting with the code.



The screenshot shows a web browser window titled "C:/Projects/Intro-Programming-AI/Conditions-Loops/speed-eval.html". The application is titled "Speed Evaluator" and features a red car icon. It has a text input field labeled "Enter Speed (km/h):" containing "99.999". A green button labeled "Evaluate Speed" is below it. A yellow box displays the message "99.999 km/h is AVERAGE!" with a blue car icon. At the bottom, a section titled "Speed Categories:" lists categories: "Fast: 100 km/h or more", "Average: 40-99 km/h", "Slow: 1-39 km/h", and "Invalid: 0 or negative values".

SWITCH-CASE STATEMENT



Switch-Case Statement

Checking Multiple Conditions Together

The "switch-case" Statement

- **Switch-case** works like series of **if-else-if-else-...**
- Tests an **expression** and does different thing for each **matched case**
- Put "**break**" or "**return**" at the end of each case!
 - Prevents **fall-through**

```
switch (expression) {  
    case value1:  
    case value2:  
        // some code ...  
        break;  
    case value3:  
        // other code ...  
        break;  
    default:  
        // final code ...  
}
```

Switch-Case – Example

- Write a function to return **the day of week** by day number

```
function dayOfWeek(dayNumber) {  
    switch (dayNumber) {  
        case 1: return "Monday";  
        case 2: return "Tuesday";  
        // TODO: check all other days  
        case 7: return "Sunday";  
        default: return "Error!";  
    }  
}
```

```
console.log(  
    dayOfWeek(3));  
// Wednesday
```

```
console.log(  
    dayOfWeek(-1));  
// Error!
```

Judge link: <https://alpha.judge.softuni.org/contests/control-flow-logic/5271>

Nested if-else Statements

Using if-else Inside Another if-else

```
if (A)
{
    if (B)
        else { }
    else { }
```

Nested If-Else Statements

```
if (condition1) {  
    console.log("condition1 is met");  
  
    if (condition2) {  
        console.log("condition2 is met");  
    } else {  
        console.log("condition2 is not met");  
    }  
}
```

Nested **if-else** statement

Problem: Pricing Table

- Write a function **getPrice(product, location)** to return a product price according to this **pricing table**:

| Location | coffee | water | sweets | peanuts |
|----------|--------|-------|--------|---------|
| Sofia | 0.50 | 0.80 | 1.45 | 1.60 |
| Plovdiv | 0.40 | 0.70 | 1.30 | 1.50 |
| Varna | 0.45 | 0.70 | 1.35 | 1.55 |

```
console.log(getPrice("Sofia", "water")); // 0.8  
console.log(getPrice("Varna", "sweets")); // 1.35  
console.log(getPrice("London", "water")); // undefined
```

Solution: Pricing Table

```
function getPrice(location, product) {  
    if (location == "Sofia") {  
        if (product == "coffee") {  
            return 0.50;  
        } else if ...  
        // TODO: check the other products  
    }  
    else if (location == "Plovdiv")  
    else if (location == "Varna") {  
        // TODO: check the other towns  
        return undefined;  
    }  
}
```

CHAT + ⏪ ⚙️ ⋮ [] ×

0 Pasted Image X

JS function getPrice(location, product) { +

Write a function getPrice(product, location) to return a product price according to attached table.
Use nested if-else.

Examples:

```
console.log(getPrice("Sofia", "water")); // 0.8  
console.log(getPrice("Varna", "sweets")); // 1.35  
console.log(getPrice("London", "water")); // undefined
```

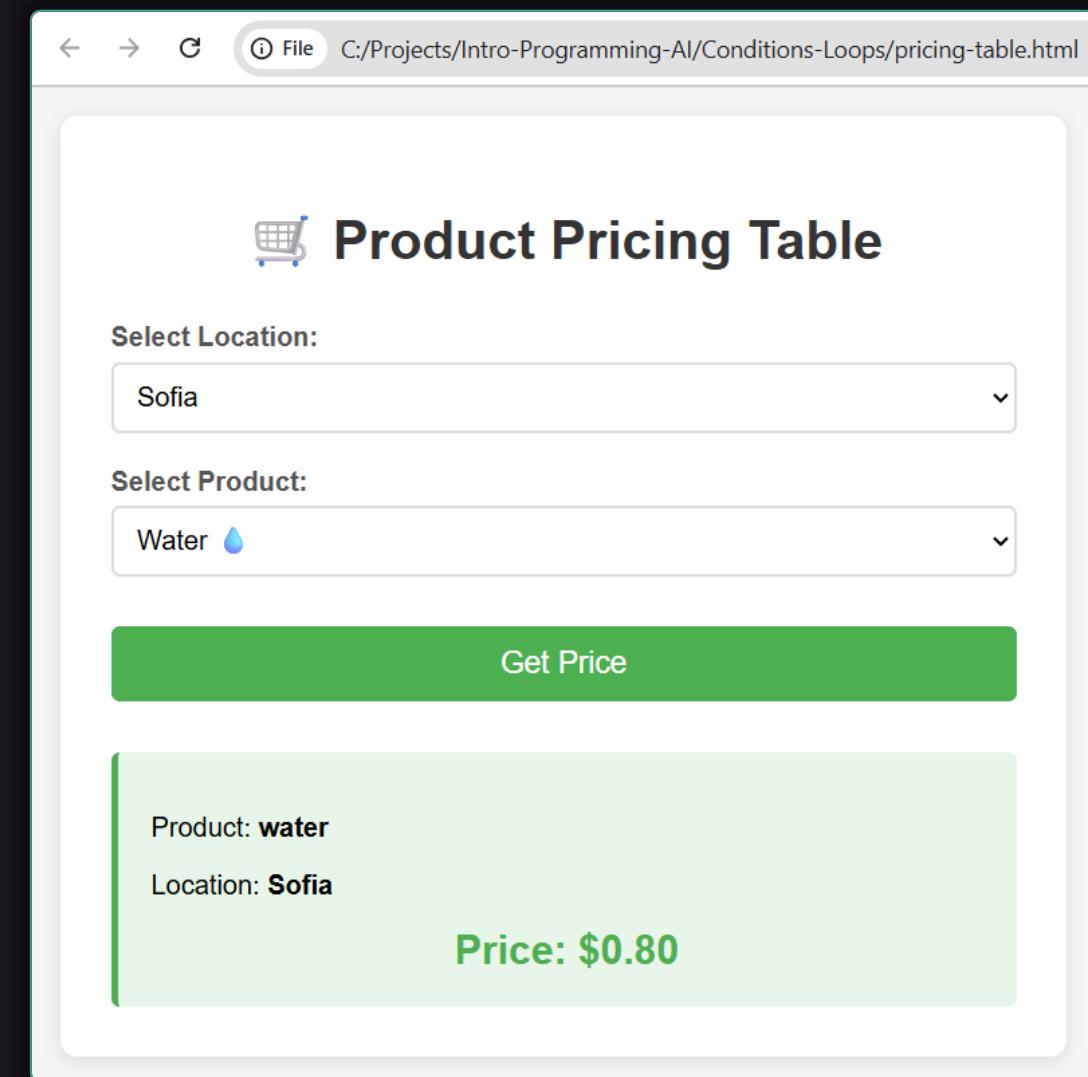
Agent ▾ Auto ▾ ✖ ➤ ▾

Judge link:
<https://alpha.judge.softuni.org/contests/control-flow-logic/5271>

Visualize as HTML

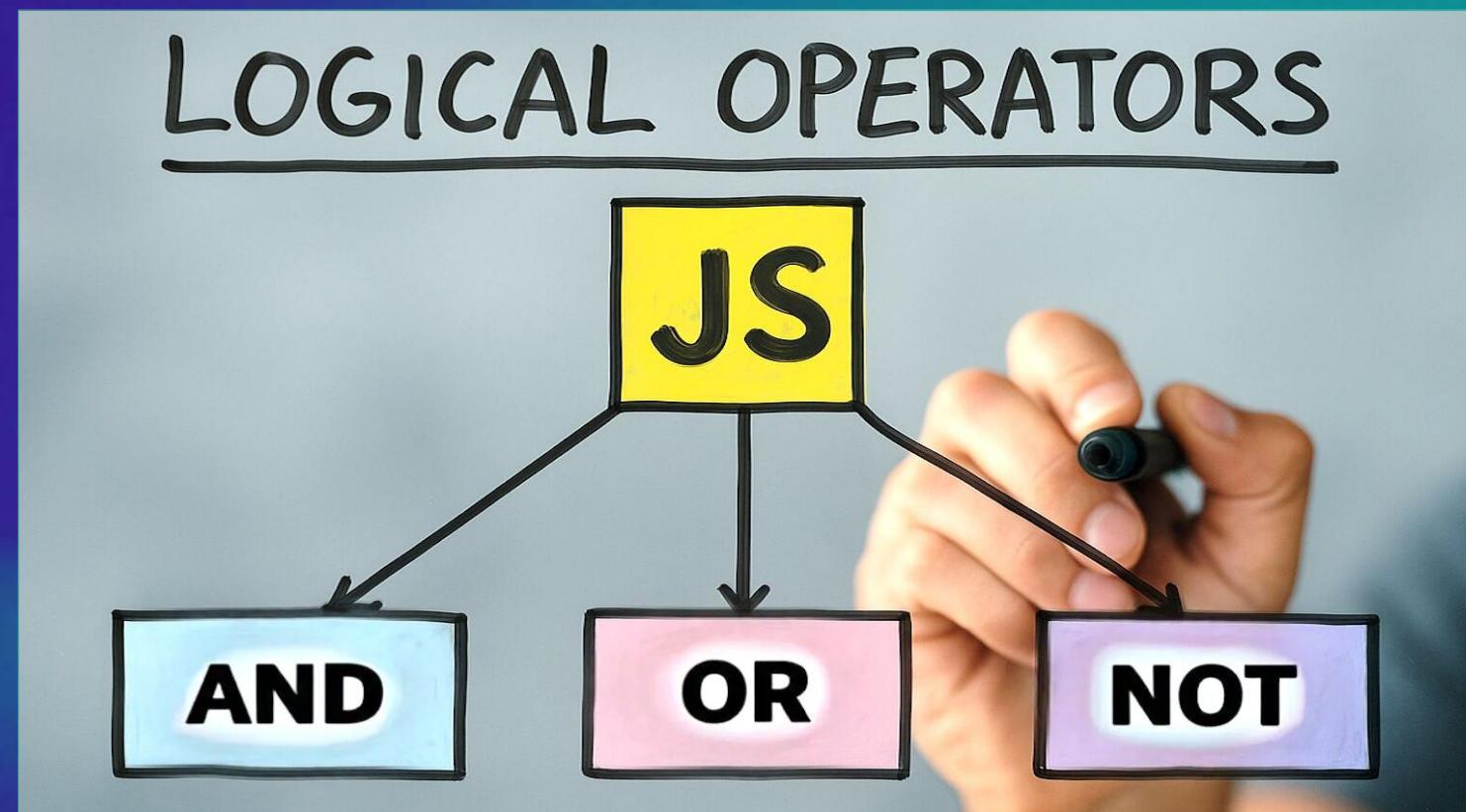
- Run a simple **Copilot prompt:**

Visualize the function
getPrice(location, product) function as HTML
page: select location
(dropdown, holding Sofia, Plovdiv, Varna) and
product (dropdown holding coffee, water, sweets, peanuts), get the price
and display it.



Logical Operators

Using Logical AND, OR, NOT and ()



Logical Operators in JS

| Action | Operator | Example |
|-------------|----------|--|
| Logical AND | && | <code>size > 0 && size < 20</code> |
| Logical OR | | <code>size < 0 size > 20</code> |
| Logical NOT | ! | <code>! (size > 0)</code> |
| Brackets () | () | <code>(age < 18 age > 60) && (price > 10)</code> |



Logical Expressions – Examples

- Logical **AND**

```
if (location == "Sofia" && product == "water")
    price = 0.80;
```

- Logical **OR**

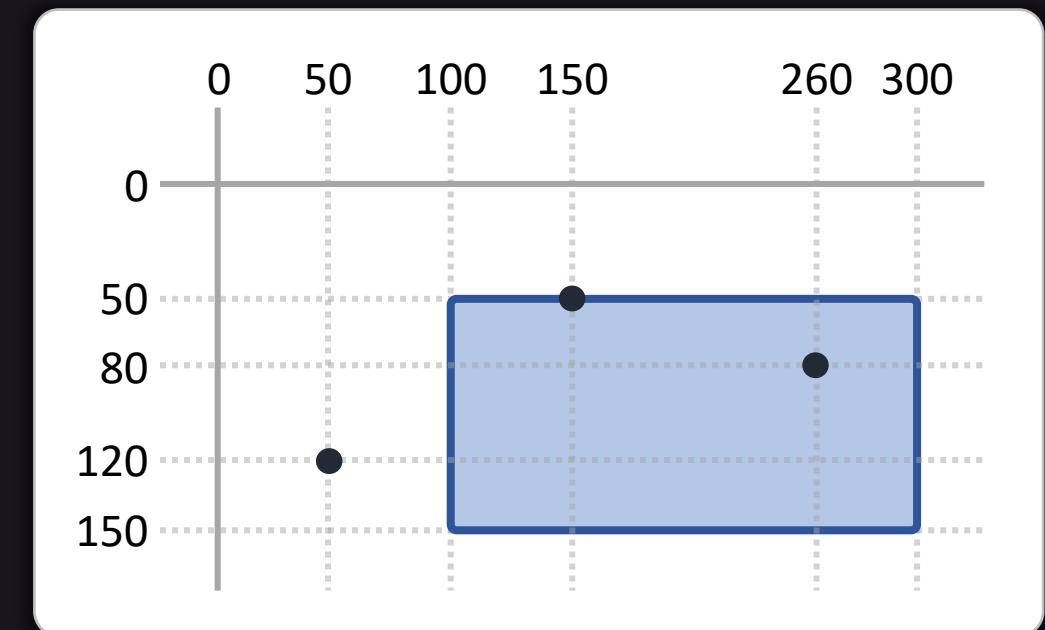
```
if (age < 18 || age > 60)
    discount = 0.20; // 20% discount for kids and retired
```

- Logical **NOT**

```
if (!(location == "Sofia" || location == "Varna"))
    price = undefined; // Invalid location
```

Problem: Point and Rectangle

- We are given a **rectangle** $\{\text{top}, \text{left}\} - \{\text{bottom}, \text{right}\}$ on the coordinate plane ($\text{top} < \text{bottom}$, $\text{left} < \text{right}$)
- We are given a **point** $\{\text{x}, \text{y}\}$
- Identify where the point lays:
 - **Inside** the rectangle
 - **Outside** of the rectangle
 - On the rectangle **border**



Point and Rectangle – Examples

```
console.log(pointOnRect(  
    50, 100, 150, 300, 50, 120));
```

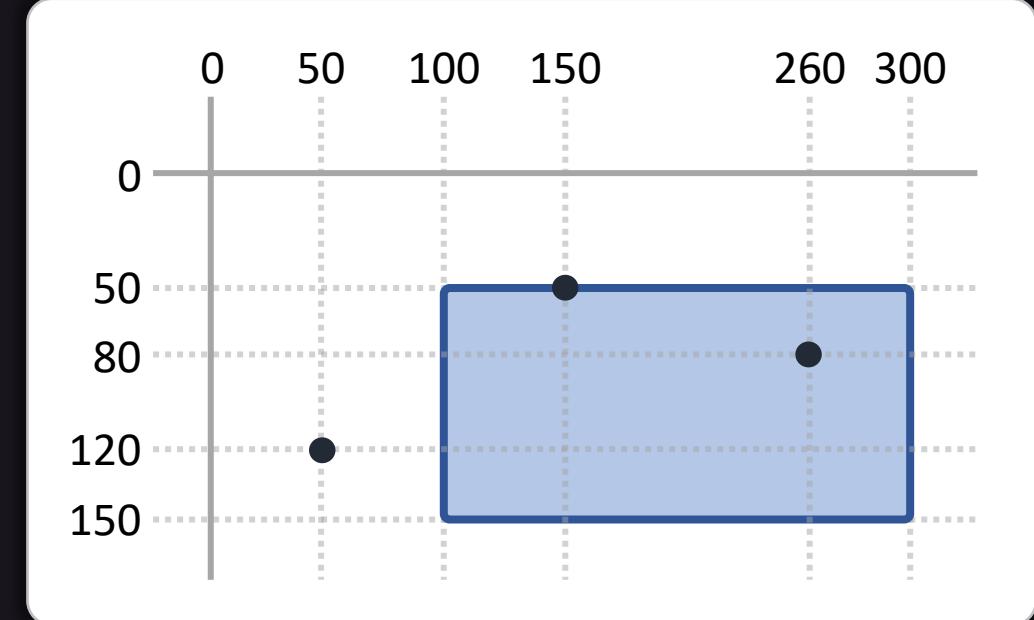
outside

```
console.log(pointOnRect(  
    50, 100, 150, 300, 260, 80));
```

inside

```
console.log(pointOnRect(  
    50, 100, 150, 300, 150, 50));
```

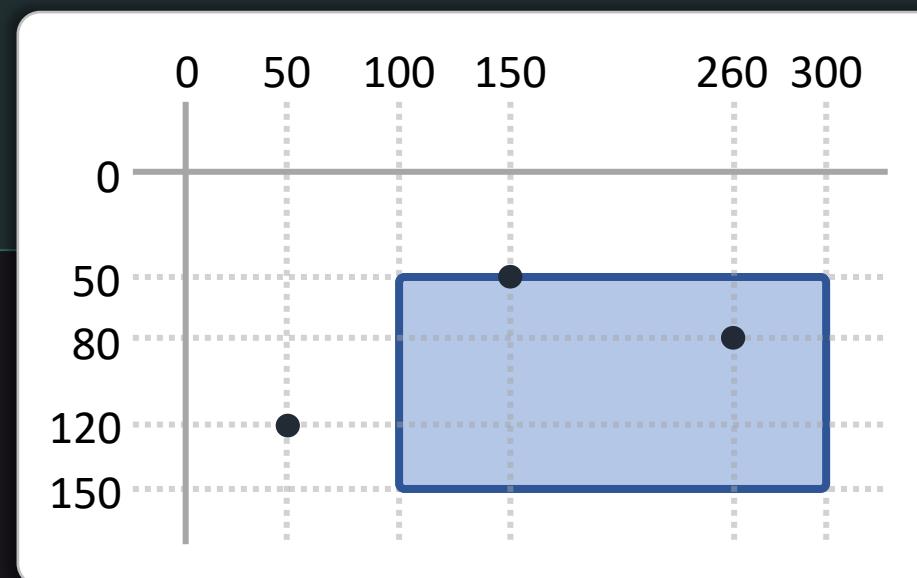
border



Solution: Point and Rectangle

```
function pointOnRect(top, left, bottom, right, x, y) {  
    if ((x > left) && (x < right) && (y > top) && (y < bottom))  
        return "inside";  
    if ((x < left) || (x > right) || (y < top) || (y > bottom))  
        return "outside";  
    return "border";  
}
```

Judge link:
<https://alpha.judge.softuni.org/contests/control-flow-logic/5271>



Visualizing the Point and Rectangle



Create a HTML page to enter rectangle coordinates (top, left, bottom, right) and point coordinates (x, y).

Display where the point lays relative to the rectangle (using the existing functions).

Visualize the rectangle with the point on the coordinate system.

JS point-on-rect.js +

Create a HTML page to enter rectangle coordinates (top, left, bottom, right) and point coordinates (x, y).

Display where the point lays (using the existing functions).

Visualize the rectangle with the point on the coordinate system.

Agent ▾ Auto ▾



Point and Rectangle – HTML Visualization

File C:/Projects/Intro-Programming-AI/Conditions-Loops/point-on-rect.html

Point on Rectangle Checker

Input Coordinates

Rectangle

Top:

Left:

Bottom:

Right:

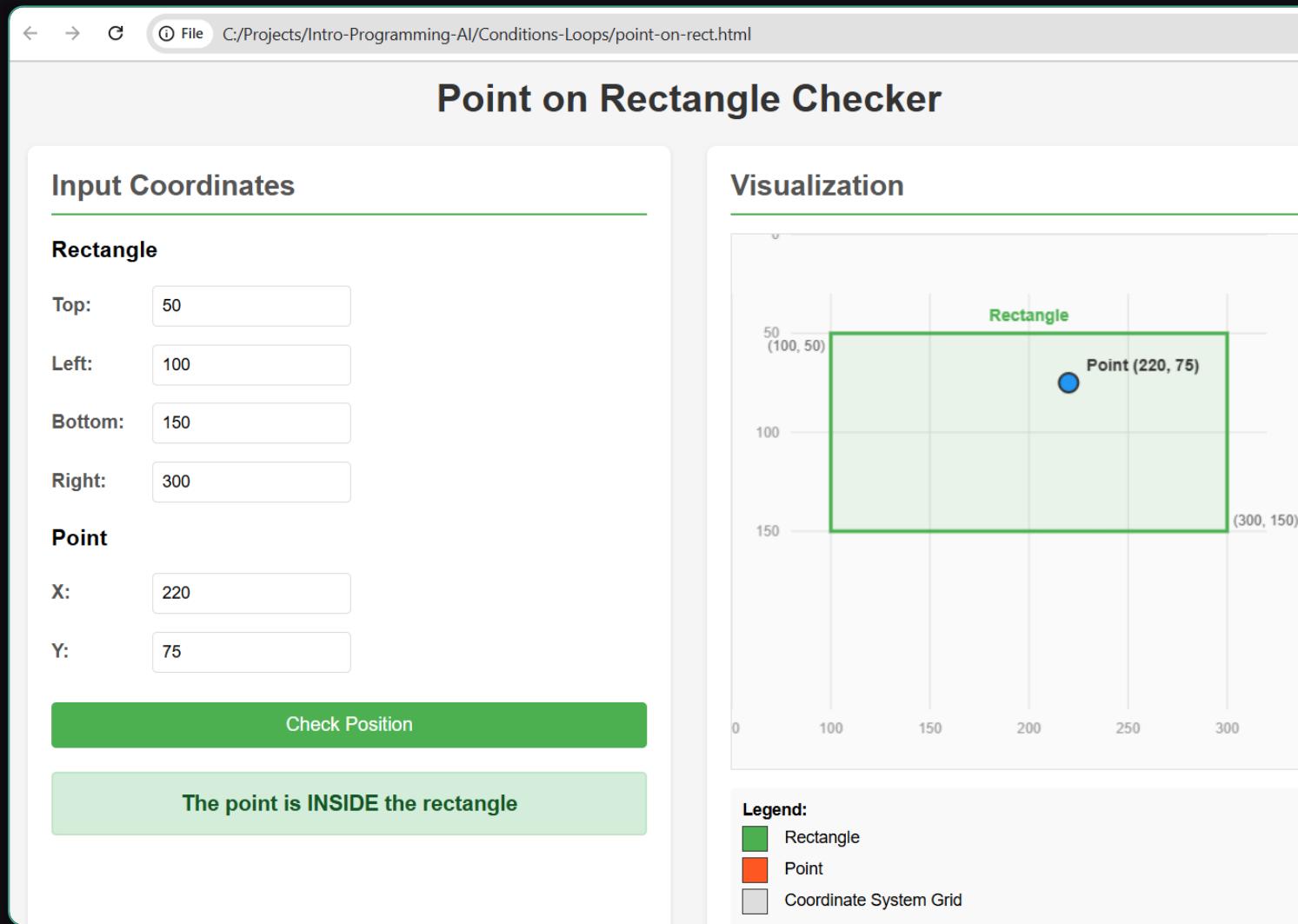
Point

X:

Y:

Check Position

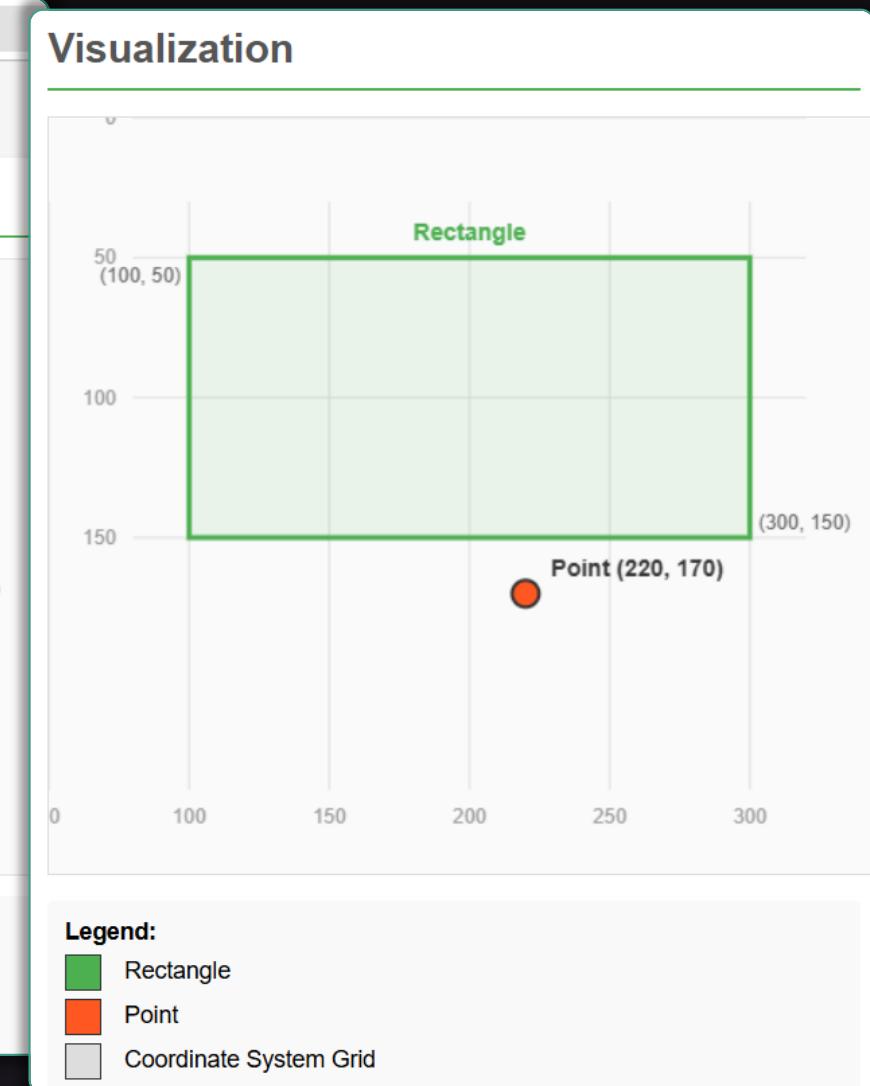
The point is INSIDE the rectangle



Visualization

Legend:

- Rectangle
- Point
- Coordinate System Grid



More If-Else Exercises

Building a Tile Arrangement Calculator

C:/Projects/Intro-Programming-AI/Conditions-Loops/arrange-tiles.html

Tile Arrangement Calculator

Enter Dimensions

Floor Width: cm
Floor Height: cm
Tile Width: cm
Tile Height: cm

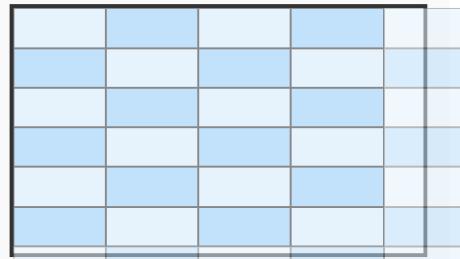
Calculate Arrangement

Optimal arrangement is vertical: 33 tiles

Horizontal Arrangement
Total tiles: 35
Layout: 7 rows × 5 columns
Tile orientation: 70 × 30 cm

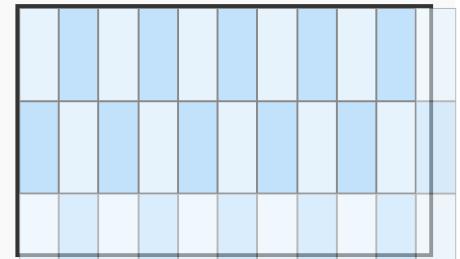
Vertical Arrangement
Total tiles: 33
Layout: 3 rows × 11 columns
Tile orientation: 30 × 70 cm (rotated)

Horizontal Layout



Border shows floor area. Tiles may extend beyond to be cut.

Vertical Layout



Border shows floor area. Tiles may extend beyond to be cut.

Rounding Numbers

- **Rounding** to the closest integer:

```
let round = Math.round(7.52); // round = 8
```

- **Rounding up** to the next integer:

```
let up = Math.ceil(23.05); // up = 24
```

- **Rounding down** to the previous integer:

```
let down = Math.floor(45.67); // down = 45
```

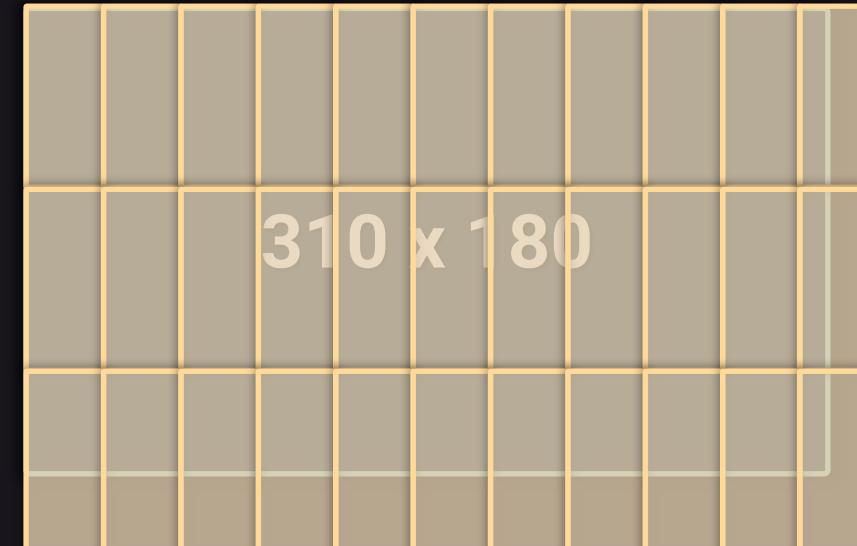
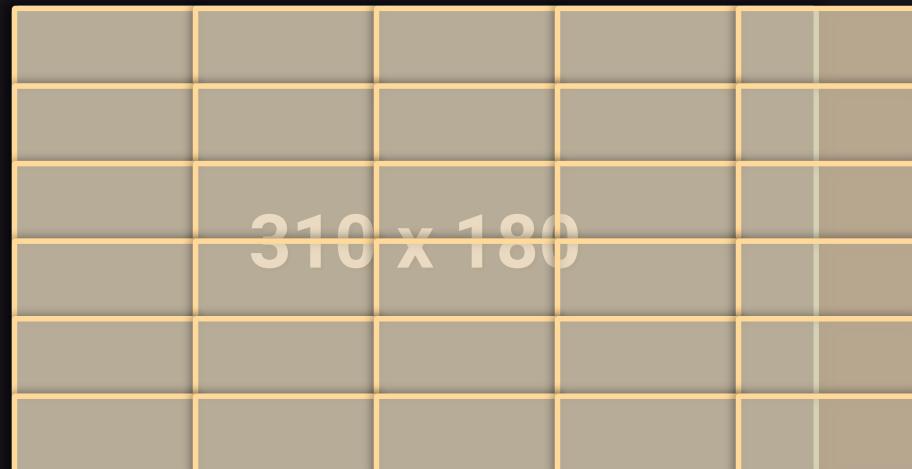
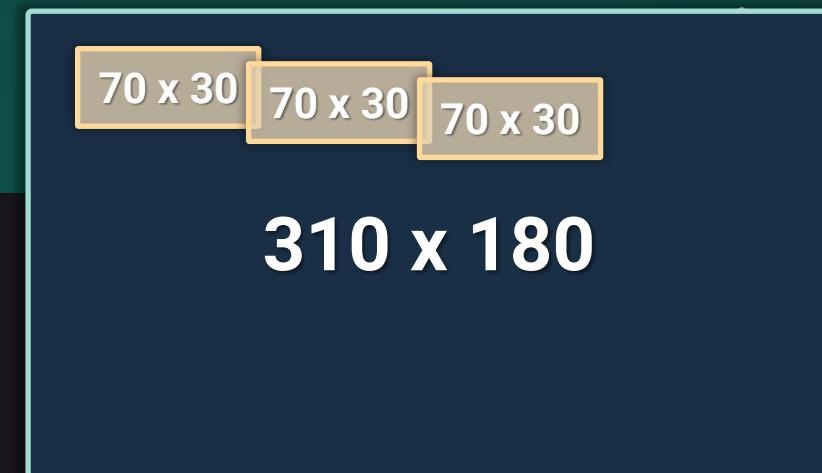
- **Format** to 2 digits after the decimal point (with rounding):

```
(123.4568).toFixed(2); // 123.46
```



Arranging Tiles

- We want to fill a floor with tiles
 - Example: floor **310 x 180**, tile **70 x 30**
 - **Horizontal** layout: **30** tiles (6 rows x 5 cols)
 - **Vertical** layout: **33** tiles (6 rows x 11 cols)



Problem: Arranging Tiles

- We have a **rectangular floor** of size **floorW x floorH**
- We have **tiles** of size **tileW x tileH**
- We want to fill the floor with **minimum tiles**
- We can **cut some of the tiles** to fit on the floor
- We can put the tiles either **horizontally** or **vertically**
- Write functions to calculate the tiles needed:
 - **calcHorizontalTiles(floorW, floorH, tileW, tileH) → {tiles, rows, cols}**
 - **calcVerticalTiles(floorW, floorH, tileW, tileH) → {tiles, rows, cols}**

Problem: Arranging Tiles (2)

- Write a function **arrangeTiles(...)** to print the **optimal tile arrangement** info for given **floor size** and **tile size**



```
arrangeTiles(310, 180, 70, 30)
```

Optimal arrangement is horizontal: 30 tiles
Horizontal: 30 tiles (6 rows x 5 cols)
Vertical: 33 tiles (3 rows x 11 cols)



```
arrangeTiles(310, 185, 70, 30)
```

Optimal arrangement is vertical: 33 tiles
Horizontal: 35 tiles (7 rows x 5 cols)
Vertical: 33 tiles (3 rows x 11 cols)

Solution: Arranging Tiles

```
function arrangeTiles(floorW, floorH, tileW, tileH) {  
    function calcHorizontalTiles(floorW, floorH, tileW, tileH) {  
        const cols = Math.ceil(floorW / tileW); // Round up  
        const rows = Math.ceil(floorH / tileH); // Round up  
        const tiles = cols * rows;  
        return { tiles, rows, cols };  
    }  
    function calcVerticalTiles(floorW, floorH, tileW, tileH) {  
        // Implement it the same way ...  
    }  
    // TODO: implement the rest of the function ...  
}
```

Solution: Arranging Tiles (2)

```
function arrangeTiles(floorW, floorH, tileW, tileH) {  
    let hor = calcHorizontalTiles(floorW, floorH, tileW, tileH);  
    let vert = calcVerticalTiles(floorW, floorH, tileW, tileH);  
    if (hor.tiles <= vert.tiles) {  
        console.log(`Optimal arrangement is horizontal:  
        ${hor.tiles} tiles`);  
    } else {  
        console.log(`Optimal arrangement is vertical:  
        ${vert.tiles} tiles`);  
    }  
    // TODO: print the horizontal and vertical calculations  
}
```

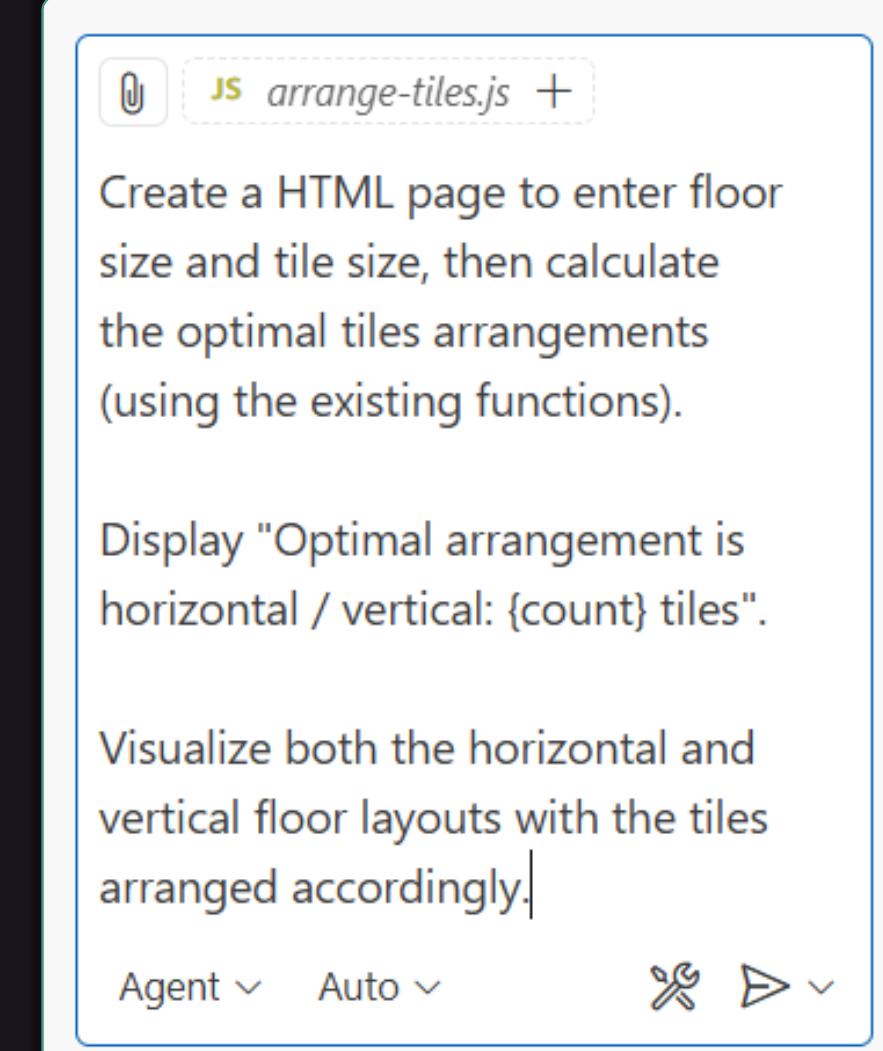
Judge link: <https://alpha.judge.softuni.org/contests/control-flow-logic/5271>

Visualizing the Tiles Arrangement

Create a HTML page to enter floor size and tile size, then calculate the optimal tiles arrangements (using the existing functions).

Display "Optimal arrangement is horizontal / vertical: {count} tiles".

Visualize both the horizontal and vertical floor layouts with the tiles arranged accordingly.



The screenshot shows a code editor interface with a file named "arrange-tiles.js" open. The code in the editor matches the text provided in the slide. Below the code editor, there are navigation buttons for "Agent" and "Auto" modes, and a set of standard file operations buttons (copy, paste, delete, etc.).

```
JS arrange-tiles.js +  
  
Create a HTML page to enter floor  
size and tile size, then calculate  
the optimal tiles arrangements  
(using the existing functions).  
  
Display "Optimal arrangement is  
horizontal / vertical: {count} tiles".  
  
Visualize both the horizontal and  
vertical floor layouts with the tiles  
arranged accordingly.
```

Agent ▾ Auto ▾ ⌂ ⌂ ⌂

Visualize the Tiles Arrangement (2)

← → ⌂ File C:/Projects/Intro-Programming-AI/Conditions-Loops/arrange-tiles.html

Tile Arrangement Calculator

Enter Dimensions

Floor Width:
310 cm

Floor Height:
185 cm

Tile Width:
70 cm

Tile Height:
30 cm

Calculate Arrangement

Optimal arrangement is vertical: 33 tiles

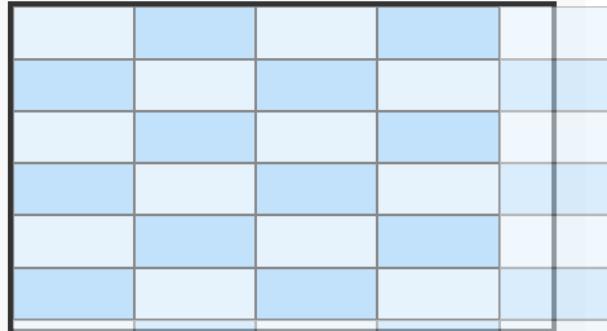
Horizontal Arrangement

Total tiles: 35
Layout: 7 rows × 5 columns
Tile orientation: 70 × 30 cm

Vertical Arrangement

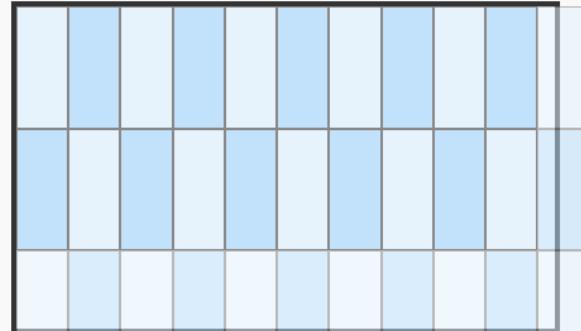
Total tiles: 33
Layout: 3 rows × 11 columns
Tile orientation: 30 × 70 cm (rotated)

Horizontal Layout



Border shows floor area. Tiles may extend beyond to be cut.

Vertical Layout



Border shows floor area. Tiles may extend beyond to be cut.

Postbank – Exclusive Partner for SoftUni AI



- One of the leading **banking institutions** in Bulgaria
- Member of the Eurobank Group with € 99.6 billion of assets
- Innovative trendsetter in the digitalization and transformation
- Certified Top Employer 2024 by the international Top Employers Institute
- AI integration for business, learning and development
- AI Assistants for talents: Story Builder, CV Assistant, Interview Trainer
- Proven people care and wellbeing initiatives
- Benefits and unlimited access to professional, personal and leadership trainings and programs
- www.postbank.bg / careers.postbank.bg



Diamond Partners of Software University



**SUPER
HOSTING
.BG**



Diamond Partners of SoftUni Digital



**SUPER
HOSTING
.BG**



Diamond Partners of SoftUni Digital



HUMAN

IMPULSE MEDIA®



1FORFIT

NETPEAK
DIGITAL GROWTH PARTNER

ABC DESIGN &
COMMUNICATION

Zahara
dig.it.all

ETIEN YANEV
Break Your Limits. Live Your Brand.



Diamond Partners of SoftUni Creative



**SUPER
HOSTING
.BG**



Organization Partners of SoftUni Creative



dequitas.

Фигма́йстор

дизайнът
на Нещата



THE
BUCKS
TOWN'S
WORK